Checkout and Launch Control System

Real Time Control Application Software Implementation Standards and Guidelines

84K01720
Draft 3
6 April 1998

RTC Applications Software Implementation Standard

Approval :

Benjamin Bryant NASA CLCS Applications Software	James Hurst USA CLCS Applications Software

Concurrence :

Richard Ikerd USA RTC Application Software Product Team	Ken Hale NASA RTC Application Software Product Team

Prepared By : Real Time Control Applications Software
United Space Alliance
Kennedy Space Center, Florida

Supporting Document Note : Acronyms and definitions of many common CLCS terms may be found in the following documents: CLCS Acronyms 84K00240 and CLCS Project Glossary 84K00250.

REVISION HISTORY

REV	DESCRIPTION	DATE

LIST OF EFFECTIVE PAGES				
Dates of issue of change pages are:				
Page No.	**A or D***	**Issue or Change No.**	**CR No.**	**Effective Date****

TABLE OF CONTENTS

1. INTRODUCTION

Consistent programming style is essential to improve maintainability, portability and to reduce programming errors. Checkout and Launch Control System (CLCS) project level standards and guidelines provide overall direction for this consistency. This standard provides rules and recommendations that address issues pertaining to the implementation of Real Time Control (RTC) Application Software.

1.1 PURPOSE

The purpose of this document is to define the style of implementation for the major elements of the RTC Application Software architecture. This collection of standards and guidelines should be viewed as a living document. It will be updated as required.

All software shall adhere to the current revision of this document at the time of software development. Revisions to the standards document shall not drive software updates unless explicitly directed.

1.2 SCOPE

This standard applies to all RTC Application Software developed or maintained (not including COTS software) for use in the Space Shuttle program at Kennedy Space Center. The standard covers implementation style. Program design and architecture are beyond the scope of the document.

This document is a complement to the following documents:

- 84K07500-010 Programming Standards Document - This is the project level document that provides standards and guidelines of a language area. These areas will not be addressed by this Implementation Standard.

- 84K00230 CLCS HCI Style Guide and Standards - This is the project level document that provides generic guidelines and standards for the CLCS project. This Implementation Standard builds upon the concepts of the 84K00230 document to provide a common RTC Application Software interface implementation.

- 84K01710 - RTC Application Software Architecture Standard - This is the document that defines the common design/architecture that will be used as the starting point for all RTC Application Software products.

This document contains both rules and recommendations which are clearly defined. Sections designated as rules must be adhered to, unless an exception is specified in this document. Recommendations are provided as a guide for developing all software in as common a style as possible without imposing unnecessary restrictions. Adherence to the recommendations is suggested and encouraged, but is not mandatory. Rationale for non-compliance with a recommendation shall be documented in the associated Software Design Specification (overview or detailed).

1.3 AUTHORITY

This document is controlled by the Checkout and Launch Control System (CLCS) Application Software Chief or the appointed representative.

1.4 STANDARDS MODIFICATION

To request new rules/recommendations or changes to the rules/recommendations of this document, a Razor issue against the document must be submitted. All issues submitted will be reviewed and dispositioned by the RTC Application Software Product team or an appointed representative.

1.5 REFERENCE DOCUMENTS

The following documents were used in the development of or are referenced in this standard:

84K00070-200	RTC Application Software Development Plan
84K07500-010	Programming Standards Document
84K00230	CLCS HCI Style Guide and Standards
84K01710	RTC Application Software Architecture Standard
84K01730-101	RTC Application Software Test Application Display Development
KSC-STD-SF-0004B	Safety Standard for Ground Piping Systems Color Coding and Identification

1.6 WAIVERS AND EXCEPTIONS

There may arise circumstances where a CSCI cannot adhere to the standards defined by this document due to an operational requirement. Waivers and exceptions to the standard may be requested by a CSCI by following the waiver process as defined in 84K00070-200 RTC Application Software Development Plan.

Approved exceptions and waivers are listed below:

Number	CSCI	Description

2. USER INTERFACE STANDARDS

This section defines the rules and recommendations that are associated with Test Application Displays used to provide data representation and command/control mechanisms.

2.1 DISPLAY ATTRIBUTES

This section describes the common attributes of a RTC Application Software Test Application Display.

Rule 2-1 : Each Display shall have a title that identifies the name and revision of the Display. Note : For Sherrill-Lubinski Graphical Modeling System (SL-GMS™) generated displays, the Test Application Display Driver implements this rule.

Recommendation : Display the title in the window frame. This will free up more space in the actual display.

Rule 2-2 : Icons representing a Test Application Display shall include the display name or a reasonable abbreviation of the name with the icon. Note: For SL-GMS generated displays, the Test Application Display Driver implements this rule.

Recommendation : Data path health status (e.g. Gateway or HIM status) should not be duplicated on a Test Application Display.

All gateway/HIM/format health status and information for a CLCS Test Set is provided on the Command Navigation System (CNS) which is always visible on a Command and Control Workstation (CCW).

Recommendation : Each Test Application Display should have an activity indicator that is located in the upper right corner of the display to indicate the software is updating.

Recommendation : Display size and resizing options should be specified by the responsible IPT.

2.2 COLOR USAGE

This section defines the use of colors in Test Application Displays. In an effort to maintain consistency across all CSCIs in a CLCS Test Set and to enhance safety, a set of colors has been standardized. Use of these colors is limited to the conditions described. Use of all other colors is at the discretion of the implementing IPT and shall be documented in the associated CSCI Overview Software Design Specification Document.

The conditions listed in the following sections may also be shown graphically by unique symbols for the type of component and condition being displayed. The symbol usage shall be consistent throughout the CSCI and CLCS Test Set. The symbols should follow the same general color scheme for the condition being displayed if it does not adversely affect the symbol look.

The following table defines the Red-Green-Blue (RGB) values for the primary colors defined in this standard. These values are defined in the standard *colordef.dat* file used by SL-GMS located in the Common Application Support Repository.

Colordef ref no.	***Color***	***R***	***G***	***B***
6	Cyan	14	245	235
58	Green	0	255	0
1	Red	234	0	0
3	Yellow	255	255	0
79	Magenta	255	0	255
0	White	255	255	255
7	Black	0	0	0

2.2.1 Normal Condition

Data in a normal condition is defined as data that is valid and within the accepted limits or states. Normal data (displayed as text) shall be displayed as follows.

Rule 2-3 : Normal condition data shall be displayed in ***cyan*** text on any background.

Exception : If it is desired to show the difference between inactive/unpowered data and active/powered data, the inactive/unpowered data should be displayed in ***cyan*** and the active/powered data should be displayed in ***green***.

Rule 2-4 : Transition - discrete data in this condition (e.g. a valve moving from CLOSED to OPEN) shall be displayed in ***magenta*** text on any background. ***Magenta*** identifies physical properties or movement. Analog data should never be considered “in transition”.

Recommendation : Labels should be displayed in a different color than data to avoid potential confusion.

2.2.2 Error/Warning Condition

Data in an error or warning condition is defined as data that is valid but is outside the accepted limits/state or represents a condition that requires immediate operator attention.

Rule 2-5 : Error/Warning conditions (displayed as text) shall be displayed using any of the following color combinations:

- ***Red*** text on any background
- ***White*** text on a solid ***red*** background

2.2.3 Caution Condition

Data in a caution condition is defined as data that is valid but is approaching the accepted limits/state or represents a condition that requires special attention by the operator.

Rule 2-6 : Caution conditions (displayed as text) shall be displayed using any of the following color combinations:

- ***Yellow*** text on any background
- ***Black*** text on a solid ***yellow*** background

2.2.4 Invalid Data

Invalid data is defined as data whose health status has been flagged as invalid by the CLCS Data Distribution Processor, which may mean the data source has failed (e.g., Gateway, HIM), the measurement is not supported by the current OI/GPC format, the data path (e.g. power supply, signal conditioner, etc.) has a break in it or the data has been bypassed by the operator or Application Software.

Rule 2-7 : Invalid data (displayed as text) shall be displayed using any of the following color combinations:

- ***White*** text on any background except for ***red***
- ***Black*** text on a solid ***white*** background

Recommendation: At the discretion of the user, invalid data can be removed from the screen by making the data point invisible. This should only be done via manual request of the user. The data shall be automatically re-displayed when it becomes valid again.

2.2.5 Common Fluid Colors

To provide a consistent look across all CSCIs in a CLCS Test Set, the following set of colors has been identified for use when depicting fluids. Note : The KSC-STD-SF-0004B Safety Standard for Ground Piping Systems Color Coding and Identification colors are provided as a reference. Where possible, the colors to be used for RTC Application Software Test Application Displays mirrors this standard. Deviations from the pipe color standard are made to provide a marked distinction between fluids that might have a common color in the KSC-STD-SF-0004B standard.

Rule 2-8 : The following colors shall be used when displaying fluids:

Commodity	***KSC-STD-SF-0004B***	***Test App Color***	***Colordef Ref No.***	***R***	***G***	***B***
Air	Green	LightBlue	43	189	209	237
Ammonia	Brown	LightPink	80	255	181	194
Argon	Gray	MediumPurple	88	148	112	219
Freon	Gray	Aquamarine	48	128	255	212
Helium	Gray	Wheat	37	204	186	150
Hydraulic Fluid	Yellow	Coral	25	249	128	115
Hydrazine	Brown	DarkOrange	75	255	145	0
Hydrogen	Yellow	LightYellow	19	244	146	180
Nitrogen	Gray	Gray	15	122	122	122
Nitrogen Tetroxide	-	RustBrown	26	140	36	37
Oxygen	Green	LightGreen	10	255	171	152
Water, Firex	Red	MediumBlue	41	0	0	204
Water, Potable	White	CadetBlue	47	94	158	160

2.2.6 Miscellaneous Color Usage

Rule 2-9 : All displays shall be created using a background color on which all of the data shall be visible, regardless of condition. Data may be displayed in different colors depending on factors such as data source, error condition, etc. All ranges of the colors that the Display requires shall be visible on the background.

Recommendation : To differentiate between measurement sources (i.e. downlist, downlink, LDB), the border or background color of the associated text box can be modified. A legend should be provided to indicate the meaning of border colors used in this manner.

2.3 SYMBOLS

A collection of graphic symbols and components needed to develop a Test Application Display is located in the Common Application Support (CAS) Repository.

Rule 2-10 : The components in the CAS Repository shall be used if the required functionality is satisfied by the component.

Each IPT will define the graphic symbols and functions required to support their Displays. All new symbols must be developed following the graphical component process as described in 84K01730-101 RTC Application Software Test Application Display Development.

2.4 ANIMATION

Animation or dynamics within a Display, can be useful additions for a quick status indication. Examples of animation include dynamically filing tanks or pipes, moving needles on gauges and changing colors on Display graphics. Animation related performance issues should be addressed by the associated IPT during the Display design phase.

Rule 2-11 : The use of animation shall be limited to areas that provide a distinct functionality. Frivolous use of animation shall be avoided.

2.5 CONTROL INTERFACES

Control interfaces are the methods used by an operator to initiate actions from a Display. These actions include initiating hardware activity as well as initiating/controlling another software process. Control interfaces deal with moving the cursor on the display screen and initiating, executing, disabling or aborting the control action.

2.5.1 Cursor Control Point

Targets or “hot spots” on a Display used for initiating functions or actions are referred to as cursor control points Navigation of the cursor to the control points is achieved by movement of the associated mouse (the primary mechanism) or by the TAB key on the keyboard (if the control point has a TAB set at its location).

Activation of a cursor control point may be performed by one of two methods:

1. By positioning the cursor over the control point and pressing Mouse Button 1 (the left most button).

2. By positioning the cursor over the control point and pressing the ENTER key from the keyboard.

Control activation may be implemented in one of two ways.

1. Control may be implemented as a single step action, where the cursor is positioned over a cursor control point and is activated. The pre-defined action is taken immediately upon release of the mouse button or release of the key-press.

2. Control may also be implemented as a multi-step command as follows. When the cursor is positioned over a cursor control point and activated, a pop-up menu is displayed that contains the valid functions. Subsequent selection of a valid function from the menu will result in the pre-defined action being taken immediately upon release of the mouse button or key-press.

Rule 2-12 : Mouse initiated hardware control actions from a Display cursor control point shall be via Mouse Button 1.

Rule 2-13 : Mouse initiated software utility control actions (e.g. status FD, plot, DMON) from a Display cursor control point shall be via Mouse Button 3 (right most button).

Rule 2-14 : A cursor that enters an active cursor control point shall require additional operator input prior to invoking an action (e.g. mouse button control).

Rule 2-15 : The pop-up function menu shall have the target identifier as the menu title.

The pop-up function menu may be dismissed and all pending actions erased by activation of the Mouse Button 3 in the menu title area.

Rule 2-16 : The pop-up function menu shall have a “Disarm” option that removes the menu from the display without taking any other action.

Rule 2-17 : The pop-up function menu shall have separator bars between each of the valid function to help preclude inadvertent activation.

Rule 2-18 : The pop-up function menu shall have a time-out defined by the IPT. When the time-out expires with no function being selected, the function menu shall be removed from the Display.

2.5.2 Prompt Control

Prompts are user control interfaces between application processes that require operator input.

Rule 2-19 : All prompts shall be displayed to an user response window which provides the user with the valid responses for the prompt.

Rule 2-20 : All prompts shall be written to a level of detail required to allow for an appropriate response without requiring further operator reference.

2.5.3 Programmable Function Keys

Programmable Function Key (F1 - F12) usage is defined in the HCI Style Guide and Standards, 84K00230.

Recommendation : Whenever programmable function keys are utilized by a program, the mapping of the active keys should be displayed, or a mechanism provided to display the mapping.

2.5.4 Fixed Function Keys

The fixed function keys (e.g. PageUp, PageDown) may be used to manipulate the RTC Application Software as long as their use is not prohibited by the operating system and their interaction is defined for the operator.

Restriction : The ‘Home’ key is reserved for use with the HCI Manager Program.

2.6 CURSOR GRAPHICS

The use of different cursor shapes is a good mechanism for informing the user of the state of the software. Actual symbol usage is dependent on the platform, however it shall meet the intent of the following rules.

Rule 2-21 : The point cursor graphic shall be used under normal operation conditions.

Rule 2-22 : An hourglass or stop-watch status graphic shall be used when the software is performing a non-momentary activity. This indicates that the associated software is performing an activity that is non-interruptible.

Rule 2-23 : A steady state I-Bar graphic shall be used when the cursor is over a data input field.

2.7 MENU STANDARDS

The following rules apply to all pop-up and pull down menus associated with RTC Application Software:

Rule 2-24 : Three ellipses points (. . .) shall be used in menus and on buttons to indicate the presence of an associated display.

Rule 2-25 : An arrow shall be used to indicate the presence of an associated sub-menu.

2.8 MESSAGE NOTIFICATION

RTC Application Software messages will be color coded to identify specific characteristics of message data Messages in the context of this section means information that is displayed/recorded to the System Message Viewer. Messages displayed directly to a Test Application Display are not covered by this section.

Note : All messages that are displayed using the System Message Utility are automatically recorded for historical purposes; no special RTC Application Software action is necessary.

Rule 2-26 : All RTC Application Software messages/prompts shall include the name of the initiating program/Display.

Rule 2-27 : The color scheme for the display of messages and prompts shall as follows:

- Cyan - information message
- Yellow - Caution message
- Red Error/Warning message

Rule 2-28 : All messages and prompts shall be time-tagged with the applicable JTOY.

3. SL-GMS DISPLAYS

This section defines the rules and recommendations that are associated with the development of RTC Application Software Displays that are generated using the commercial off-the-shelf tool Sherrill-Lubinski Graphical Modeling System (SL-GMS™).

3.1 STANDARD COLOR MAP

The SL-GMS drawing tool uses a data file to define the available colors. This file has been developed and is under configuration control. It is located in the CAS Library.

Rule 3-1 : All RTC Application Software Test Application Displays developed using SL-GMS shall use the standard "*colordef.dat*" file.

3.2 SL-GMS COMPONENT AND DISPLAY STANDARDS

The following set of rules and recommendations define how SL-GMS components and displays are to be developed to ensure commonality across all systems in a CLCS Test Set.

Rule 3-2: All graphical components and displays shall be developed following the process defined in 84K01730-101 RTC Application Software Test Application Display Development.

Rule 3-3 : All graphical components shall be moved to and saved at location (0, 0).

Rule 3-4 : The double buffer, batch erase, erase and redraw flags shall be set to FALSE on a display. Use of the flags on a component level is acceptable.

Rule 3-5 : All hidden or obscured parts of a component shall be named.

Rule 3-6 : Colors used for the display of information, data or status shall not be hard-coded into an SL-GMS component. Variables shall be used (these variables must be added to the CSCI color scheme).

Rule 3-7 : Grid size shall always be a multiple of the default size (2x2) to support scaling of the component (e.g. 1x1, .5x.5, .25x.25).

Recommendation : Components should be built "snapped to grid" whenever possible.

4. CONTROLSHELL COMPONENTS

This section defines the rules and recommendations that are associated with the various RTC Application Software elements that will be developed using the commercial development tool ControlShell™. The following acronyms/definitions are used in the standards descriptions:

- EIM End Item Manager
- EIC End Item Component
- FSM Finite State Machine
- ATC Atomic Component
- STC State Transition Component
- COG Composite Object Group
- DFC Data Flow Component

Rule 4-1 : The CAS and EIM Services Repositories shall be searched for compatible reuse components prior to building a new one. Compatible components shall be reused - not copied.

4.1 END ITEM COMPONENTS

Rule 4-2 : EICs will be implemented as graphical objects (ATC, COG or FSM).

Rule 4-3 : If the design/implementation of an EIC meets any of the following criteria, the EIC shall be implemented using an FSM/COG to describe its operation:

- The EIC is required to be interruptible
- The EIC contains time delays greater than two seconds
- The EIC is composed of more than two molecules (services are not considered molecules)
- The EIC is a composite component composed of other EICs
- There are more than four command interfaces
- There are more than eight associated Function Designators (excluding HIM relay status FDs)
- The operation of the EIC is really a sequence of smaller operations

Recommendation : If the EIC design/implementation does not meet any of the above rule, the level of "drill down" to be used is at the discretion of the implementing Integrated Product Team. In the interest of reuse, when developing these EICs, the IPT should look for reusable molecules. Molecules with the potential for reuse should be placed into the CAS Library.

A molecule is the lowest level of encapsulation for a component. A molecule normally contains only Function Designators and the methods to access/use them. It does not contain other molecules or complex functions.

4.2 FINITE STATE MACHINE DIAGRAMING

The following sections describe the standards and guidelines to be followed when implementing a finite state machine (FSM) for Sequencers and EICs using ControlShell.

Note : Case sensitivity in names is a requirement of ControlShell.

4.2.1 Labeling and Names

Rule 4-4 : "Start" (case sensitive) shall be used as the initial state in a nested composite state and shall be made an entry state. This state shall be at the top or left of the page to follow a left-to-right / top-to-bottom flow.

Rule 4-5 : "Complete" (case sensitive) shall be used as the final state in a nested state and will be made an exit state. This state shall be at the bottom or right of the page to follow a left-to-right / top-to-bottom flow.

Rule 4-6 : Names, including instance names, shall start with an alpha character (no underscores, special symbols or numbers). All names shall follow the naming standards defined in 84K07500-010 Programming Standard Document.

Recommendation : Instance names of composite states should be meaningful.

Rule 4-7 : When creating a new composite state, its name (not its instance name) should accurately encompass the logic of the underlying sequence (i.e., "VerifyPrerequisites" composite state should encompass all logic associated with verifying prerequisite conditions only).

Rule 4-8 : State names should follow the C++ naming conventions defined in the 84K07500-010 Programming Standards Document.

Rule 4-9 : Where a logical reference is intended, a logical, upper case name shall be used and not a number (e.g. use ON or OFF versus 1 or 0).

Recommendation : Transition conditions shall be complete and explicit on the sequence diagram.

4.2.2 Common Elements

Recommendation : All Sequencers should contain at least one ATC which contains the top level logic. It maintains state (e.g. IDLE, RUNNING, STOPPED) and sends all top level control stimuli (e.g. run, secure, terminate). It is a CORBA server for the operator interface. This is referred to as the Sequencer Object ATC. It should be common between Sequencers or inherited from a base class, when additional functionality is required (such as unique operator input functions).

Rule 4-10 : The Sequencer Object ATC, if present, shall maintain and publish the state of the Sequencer (e.g. IDLE, RUNNING, SECURING).

Rule 4-11 : All Sequencers shall contain at least one ATC which contains the logic to send messages to the operator. Other components will use this method to send messages.

Rule 4-12 : Sequencers shall contain ATCs for reading FD information. They will provide a method "ReadData" which is connected to a DFC outside the FSM that uses periodic reads.

The ATCs will normally be placed at the Sequencer's FSM top level. They are placed there to provide a single location for each reference and to enhance reuse of lower level composite states. Note: The reason for FD level reads for periodic data rather than CORBA calls to an EIC for values is the Current Value Table reads are a magnitude of order faster than CORBA calls.

Rule 4-13 : Sequencer shall contain CORBA client ATCs for End Item functionality

These ATCs will normally be placed at the Sequencer's FSM top level. Interfaces will be passed through to the levels where it is required. At that level, a "splitter" will separate the EIC methods so that a STC can be connected to a single function. Only import methods for the splitter must be connected; all others are optional.

Recommendation : STCs associated with transitions or with entry or exit from a state shall initiate end item methods by importing them from the splitter.

Recommendation : Aborting an operation shall be accomplished through a transition at the top level FSM. If the logic is present at a lower state, a STC can send a stimulus to terminate or secure.

4.2.3 Standard Diagram Layout

Sequencers will normally be organized internally based on both functionality and aesthetics. This organization should normally mirror the following pattern.

Rule 4-14 : The sequence diagram shall flow from left to right, top to bottom. Loops may exist, but the entry and exit from them should follow this convention.

Rule 4-15 : The Sequencer Object ATC, if it exists, shall be located on the top level of the Sequencer.

Rule 4-16 : The message write ATC shall be located below the ports on the top level of the Sequencer.

Rule 4-17 : ATCs for end-item control (CORBA clients) shall be lined up in one or more rows below the associated FSM. They shall be segregated by an "End Item" annotation.

Rule 4-18 : ATCs for FD level reads shall be lined up in one or more rows below the End Item ATCs. They shall be segregated by an "FDs" annotation.

Rule 4-19 : Access to an EIC should be made available to an FSM using the "least common denominator" method. In other words, if the EIC is needed in only one FSM (within the application), then the EIC should reside within that FSM. If multiple FSMs (within the same application) need access to an EIC, then the EIC should reside at the diagram level containing all FSMs that need access to the EIC. EIC access is then provided via port connections.

Rule 4-20 : If an aborted state exists within the diagram, then "Abort" (case sensitive) shall be used. It shall be created as an exit boundary state.

Rule 4-21 : Developer notes, special handling, warnings, etc., shall be provided as annotations within the diagram.

Rule 4-22 : Instance names of ATCs, FSM composite states and signals shall be changed from the generic (ATCx/FSMx/Sigx) to a more descriptive name and shall follow the naming standards defined in 84K07500-010 Programming Standards Document.

Recommendation : Create a composite FSM state(s) when the diagram becomes complicated, cluttered or there is an opportunity for reusing a portion of the sequence.

Recommendation : The sequence (FSM) logic not depicted on a state transition should be placed into components (ATC/STC) with code reuse in mind. All FSM non-transition code should not be encapsulated into a single component for the FSM.

Rule 4-23 : Data pin names shall follow C++ class attribute naming standards as defined in 84K07500-010 Programming Standards Document.

Recommendation : ControlShell only displays the first three characters of pins, bubbles and interfaces on diagram components. Therefore, the first three characters of names for these items should be as unique as possible.

Rule 4-24 : Bubble names (methods) shall follow C++ class method naming standards as defined in 84K07500-010 Programming Standards Document.

Recommendation : The “Complete” state should be located towards the bottom right side of the FSM sequence.

Recommendation : Pins/Bubbles/Interfaces should be interconnected without cluttering the diagram. Remember, pins/bubbles/interfaces in FSMs and COGs do not need to be directly connected. If NOT directly connected, then attached signal lines must have the same name (remember ControlShell is case sensitive) if the connectors are to be virtually connected.

Note : “Provides” bubbles do not need to be connected. All other connectors (pins/interfaces/”uses” bubbles) must be connected to a signal line.

Recommendation : There is no need to have special names for boundary states. Rely on the on-line view of the diagram to highlight the STC/transition connection and to identify boundary states.

Rule 4-25 : A top level description of the FSM shall be included on the diagram as a annotation.

Recommendation : Related FSM states which should all respond or send common set stimuli should be encapsulated within an FSM level. For example, FSM leveling should not be used merely to convey FSM state association.

Recommendation : Attempt to keep diagrams legible at the 8.5 x 11.0 page size to facilitate making a hard copy of the diagram if necessary.

4.2.3.1 Top Level Composite Object Group

Recommendation : The top level COG is the single block FSM representation of all contained sequences. It shall contain the Reactive Control Sequence interface ATC (if applicable), Compatibility ATC and periodic data update DFC. These items will connect into their respective sequence block at this level.

4.2.3.2 Top Level of Sequencer

Recommendation : The FSM should have IDLE and RUNNING states. The transitions should be “Run”, “Complete”, “Secure” and “Terminate”. This can vary slightly depending on system-unique logic.

Rule 4-26 : The top level diagram shall contain the Sequencer Object, the Message Writer, ports, end item ATCs and FD ATCs.

Recommendation : When Sequencers are mutually exclusive, they can be implemented within a single FSM. If so, this implies one IDLE state and multiple RUNNING states. This can save extra thread and compatibility logic.

4.2.3.3 Intermediate Level of Sequencer

Based on the complexity of the Sequencer, this level will be a chained set of composite states which perform subtasks. These will normally start with "Check-Prerequisites" and follow through each subtask (e.g. Configure_GSE). This is an organizational level needed because most Sequencers are complex enough to span diagrams several pages in length. This level will usually have ports for data and end item interfaces which are passed on to the next level down if necessary.

Recommendation : These composite states should be designed to be reusable in different Sequencers within the same or similar systems whenever possible.

Recommendation : If a Sequencer task status is desired (for operator information), the breakdown at this level should match the task status milestones. An ATC at this level will publish an indication of the task in progress for a interface display to use. Strings should be used to avoid unique enumerations for each Sequencer or cryptic decoding which can lead to erroneous status.

4.2.3.4 Bottom Level of Sequencer

The bottom level of a Sequencer contains all of the detailed logic. It contains simple states and logic-based transitions (e.g. Press > limit) which map directly to the level of the functional requirements. Most STCs will be at this level of the Sequencer.

4.2.3.5 State Transition Components

Rule 4-27 : If an STC is an entry component to a state, it shall be placed above the state.

Rule 4-28 : If an STC is an exit component from a state, it shall be placed below the state.

Rule 4-29 : Use the ControlShell default (e.g. CSFSM_DEFAULT_STIM) transition name when the transition does NOT require any logic.

Rule 4-30 : All major logic shall be included as transition expressions.

Remember, code expressed with the transitions does not need explicit compilation, therefore make generous use of these expressions (DO NOT hide logic within ControlShell components).

Rule 4-31 : The name of any STC used as an entry component of a state shall be named <name_entry>.

Rule 4-32 : The name of any STC used as an exit component of a state must be named <name_exit>.

Recommendation : STCs related to a particular state (i.e. entry and exit STCs) should be grouped.

Recommendation : STCs should be kept to the right and as close as possible to the associated transition without cluttering the diagram.

Rule 4-33 : All STCs shall be placed as close to the element they are associated with as is practical. This helps associations to be clear, even on hard-copies.

Exception : When diagraming a loop, the STCs should be outside of the loop for clarity.

4.2.3.6 Ports

Rule 4-34 : Ports shall be lined up along the edge of the diagram. This allows descriptive names to be written immediately above and parallel to the signal line (instead of perpendicular to it).

Rule 4-35 : Ports shall be organized in the following categories : Interfaces, Methods, Data. They shall be labeled with these tags and in this order (highest level of abstraction to lowest).

4.2.3.7 Atomic Components

Recommendation : ATCs for end item control (CORBA clients) should be lined up in one or more rows below the FSM. They should be segregated by a label “End Items”.

Recommendation : ATCs for FD level reads should be lined up in one or more rows below the end item ATCs (again following the convention of highest level of abstraction to lowest). They should be segregated by a label “FDs”.

4.2.3.8 Component Reuse

Reuse is critical to minimizing maintenance costs. It is linking in the same code or module from the same file in many places or inheriting from a class where you can use its methods in addition to new unique methods

Recommendation : Sequencer Object ATCs will have to be created when there are unique input functions. These should inherit from the Sequencer Object ATC (which is in the CAS or EIS repositories).

The repository Sequencer Object ATC provides all of the standard methods. These methods are virtual and can be overridden if needed. An example of a possible need for this is when inputs need to be checked in the Run method before sending a “run” stimulus for a branch in logic.

Recommendation : As new end item CORBA interfaces are defined, ATCs which are clients to these will need to be developed along with the ControlShell interface components. They should follow the style of the existing ATCs and should be added to the repository when developed. Remember, multiple implementation classes may have the same interface.

Recommendation : Always try to organize / parameterize subtask FSMs to maximize reuse. Look for commonality.

Often when the same complex end item or group of simple end items is manipulated in more than one Sequencer or complex end items are similar or have similar grouping, multiple instances of the same FSM can be used.

Recommendation : Avoid creating multiple sequences with similar functions. If the logic is the same, parameterize a reusable sequence with the correct data and method pins and create multiple instances.

4.2.3.9 Signal Lines

Rule 4-36 : Straight line segments shall be used for all signals.

Recommendation : Signal lines shall be labeled directly above the line in a horizontal run.

Recommendation : Signal lines may be connected at both ends or may be only connected at one end. Connecting at both ends provides the best clarity. Connecting a separate single line to each item to be connected and then labeling them exactly the same (ControlShell is case sensitive), provides the same functionality, but avoids “spaghetti” displays. The choice should be based on the complexity of the diagram.

5. COMMON OBJECT REQUEST BROKER ARCHITECTURE

The Common Object Request Broker Architecture (CORBA) is a specification for a standard object-oriented, distributed applications architecture. It permits the remote (by a client) invocation of a known object method regardless of the object's location. The object may be executing in the same process, another process on the same computer or remotely on another computer. It assumes no prior knowledge of platforms or languages, so it readily supports both reuse and portability. This section defines the rules and guidelines for implementation topics associated with the CORBA aspects of the RTC Application Software architecture.

5.1 INTERFACE DEFINITION LANGUAGE

An Interface Definition Language (IDL) file describes the data types, operations and objects that a CORBA client can use to make a request and that a CORBA server must provide for an implementation of a given object. IDL is a definition language, not a programming language. An IDL compiler is used to produce language specific code from an IDL file. The generated code is then used when developing client and server applications. The guidelines in this section will help ensure a consistent approach to writing IDL files. Reference Appendix B for an IDL file template.

Rule 5-1 : An IDL file shall contain only one Module. The IDL filename shall match the module name.

Rule 5-2 : IDL key words shall not be used for identifier names (e.g., do not use "Sequence" as an interface name).

Rule 5-3 : C++/Java unique programming language key words shall not be used in an IDL file.

Rule 5-4 : Oneway methods shall only be used when the method is NOT modifying any data or using a method invocation which modifies any data. Oneway methods should only be used when the client is uninterested if the server method executed. The client is only assured that a write to the server socket completed successfully.

Oneway methods are considered unreliable. Oneway methods MUST have void return types and cannot contain "inout" parameters, "out" parameters or throw exceptions.

Rule 5-5 : The CLCS Programming Standard 84K07500-010 shall be followed when naming all exceptions, types, parameters, methods, enumerations and attributes. The class naming conventions shall be used for interface and module names.

Rule 5-6 : IDL files shall contain multiple inclusion protection (similar to C++ header files).

Example

```
#ifndef  [FILENAME]_IDL
#define [FILENAME]_IDL
....
#endif  //   [FILENAME]_IDL
```

Rule 5-7 : Global definitions shall not be used. All IDL declarations shall be enclosed within a module construct.

Rule 5-8 : Variable names shall be declared for all formal parameters of each method declaration :

Example : int GetTaskName (in TaskID id, out string Name) ;

Recommendation : Related interface definitions (e.g. all interfaces for a power subsystem) should be grouped together in one IDL file. Using a separate file for each interface should be avoided. The use of #include directives can support this methodology while providing maximum functionality.

Recommendation : Limit the number of parameters. This helps improve the understandability of the IDL and the performance of the method invocation. Re-factor related parameters into "structs" to help reduce the number of parameters.

Recommendation : All data types and exceptions should be defined at the module level. Allocating data types and exceptions within an interface level leads to difficulties in reuse, redundant definitions and inconsistency for developers.

Recommendation : Provide sufficient exceptions for defensive programming but do not go "over-board". Too many fine grained exceptions lead to brittle IDL, but a single exception is normally inadequate. Group exceptions into categories to avoid proliferation of definitions. Remember all IDL operations can throw system exceptions.

Recommendation : Choose data types with appropriate balance for the need for reuse with the need for inter-operability. Overly constrained data types lead to brittle IDL, but data types that are too flexible do not provide sufficient inter-operatbility.

Recommendation : The use of unions should be minimized. This will help the readability and maintainability of the code. The union IDL mapping to source code is considered complex and slow relative to performance.

Recommendation : Use inheritance within interface definitions to promote reuse and to encapsulate common functionality.

Recommendation : Use "String" instead of "Sequence <char>" for those things most naturally thought of as strings.

Recommendation : Data types are always passed by value, therefore limit the number of data type parameters in a method declaration.

Recommendation : Define attributes as "read only" unless they must be set from outside of the class definition.

Recommendation: Limit the use of attributes. Interfaces do not contain data. Attribute declarations are mapped to Get() and Set() methods for each attribute. "Read Only" attributes only map to Get() methods.

Recommendation : Limit the use of the CORBA type 'any'. Provide a set of IDL-defined data types for use with each occurrence of type 'any'. CORBA type 'any' lacks constraint, bypasses compiler checking and causes slow marshalling of communication data.

Recommendation : Use comments to describe each interface declaration. Use comment separator lines for readability.

Recommendation : When using a location service, consider using “pragma version <interface-name> #” to avoid a mismatch of client and server using inconsistent versions of the IDL. For example, a client requesting an object with a repository id = “IDL:Stocks:1.0” will not find the object reference offered by the server’s IDL:Stocks:2.1.

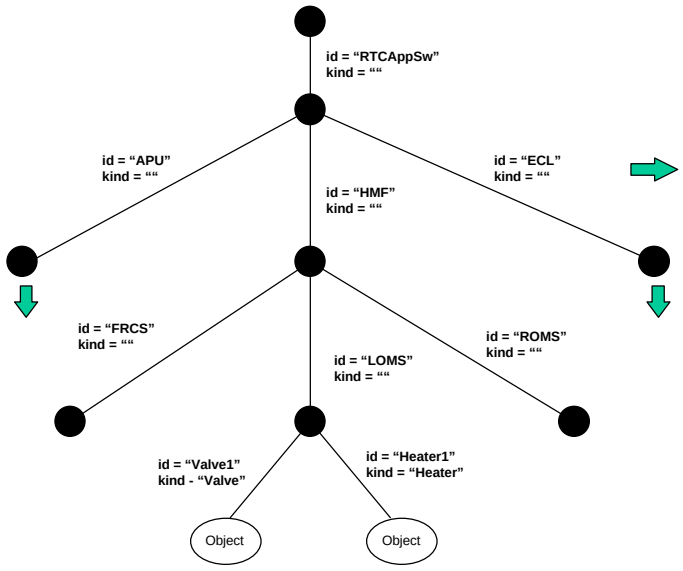
5.2 NAMING SERVICE STRUCTURE

The use of the CORBA Naming Service allows the developer to :

- Assign arbitrarily complex names to objects
- Retrieve object references using these names
- Build complex naming graphs that clearly represent the enterprise object structure
- Check for uniqueness of names
- Create an ORB independent naming scheme

The following rules and guidelines apply to the building of a Naming Service structure for RTC Application Software.

Rule 5-9 : The RTC Application Software CORBA Naming hierarchy shall be based on the project approved CSCIs (reference Appendix A) and the individual CSCs within each CSCI. The following diagram illustrates the Naming hierarchy that shall be used.



It is convenient to think of a Name as forming an edge between two nodes which can be Naming Contexts or other CORBA objects.

Rule 5-10 : The CSCI Naming Hierarchy, based on the above diagram, shall be detailed in the CSCI Overview Software Design Specification Document.

6. CONTROL LOGIC SEQUENCES

There are two types of Control Logic (CL) sequences associated with RTC Application Software: Prerequisite and Reactive.

6.1 PREREQUISITE CONTROL LOGIC

Prerequisite Control Logic (PCL) Sequences are developed for command Function Designators that are to be transmitted to end items only if a predetermined configuration exists. If the predetermined configuration does not exit, the PCL Sequence will block the issuance of the requested command.

Rule 6-1 : PCL Sequences shall only be used on a command that could cause personal injury or equipment damage. Use of PCL Sequences for software housekeeping or convenience is not permitted.

Rule 6-2 : All PCL Sequences shall be derived from the Application Services provided base class PCL Sequence. Implementation will involve defining the virtual *userExecute()* method of the base class.

Rule 6-3 : The decision whether to set the command to a potentially hazardous state or not shall be based on at least one FD other than the command that is being issued.

Rule 6-4 : All PCL Sequences shall return a numeric reason code to indicate their success or failure. A reason code of 0 shall indicate command acceptance (command can be issued by the system). A non-zero value shall be returned to indicate command rejection.

Rule 6-5 : A PCL Sequence shall not contain any time delays or logic loops. Execution shall be straight-forward and linear.

Rule 6-6 : The user code in a PCL sequence shall be limited to using Application Services API calls for the following functions only:

- Obtaining the initiating stimulus FD and its value
- Obtaining current (i.e. dynamically updated) measurement and pseudo FD values and health information

6.2 REACTIVE CONTROL LOGIC

Reactive Control Logic (RCL) sequences are developed for measurement Function Designators requiring time-critical responses for out of limits conditions. They are used to invoke a series of operations which must be performed upon occurrence of the out of tolerance condition. RCL Sequences are implemented as a sequence contained within the associated End Item Manager application.

Rule 6-7 : RCL Sequences shall be used when at least one of the following criteria is satisfied:

- Any reaction to system data requires a response time of less than one second (i.e. time from measured data to command output)
- An emergency reaction must be taken to prevent personal injury or equipment damage and that action is detectable by data provided by CLCS

Rule 6-8 : After the RCL Sequence has issued commands in response to an exception, the Sequence shall provide notification that the RCL Sequence executed via System Messages.

Rule 6-9 : An RCL Sequence shall maintain a pseudo FD representing the state of the sequence (e.g. ACTIVE, INHIBITED).

7. CONSTRAINT MANAGEMENT

This section will be provided when the CLCS Constraint Management capability is better defined.

8. MISCELLANEOUS STANDARDS AND GUIDELINES

This section contains rules and recommendations of a general nature that do not belong in any special section.

Rule 8-1 : Code modules that are not automatically generated by a COTS tools shall follow the templates provided in Appendix B.

Rule 8-2 : The audible alarm shall only be used to flag critical failures or conditions which require immediate operator action. An audible alarm shall always be accompanied by a text message describing the condition. A mechanism shall be provided to silence/disarm the audible shall.

Rule 8-3 : If GMT/JTOY is used to implement timers which are used to set up events relative to other events that occurred at an earlier time, rollover of GMT to the next day shall be accounted for to eliminate unexpected results.

GMT/JTOY roll-over is the time of day when the timers roll-over to the next day and the hours, minutes, seconds and milliseconds (for GMT) will be set to zero. GMT roll-over occurs at either 1900 EST or 2000 EDT.

Rule 8-4 : Software CSCIs shall not use Countdown Time (CDT) in their critical decision making and/or control software. Event oriented decision making and control paths (e.g. GLS milestone pseudo FDs) shall be used.

The potential exists for consoles to become out-of-sync with CDT due to timer failures or network traffic problems. Basing decision which result in the issuance of commands of a faulty CDT could lead to unanticipated hardware problems.

Rule 8-5 : In order to ensure proper communications between RTC Application Software and the on-board General Purpose Computers (GPC), the implementation of Keyboard Unit Equivalents (KBUE) commands shall be as follows:

- Ensure the correct SPEC is present
- Verify the display ID is correct prior to each item entry or series of item entries (no intervening non-KBUE commands)
- Re-verify the display ID is correct or check other data for an indication the item entry was successful after each item entry or series of item entries
- When all KBUE operations are completed, issue a RESUME command

Note : Orbiter avionics are being upgraded with Integrated Display Processors (IDP) which are replacements for the Display Electronic Units (DEU). The DEUE nomenclature in CLCS has been replaced by KBUE.

Rule 8-6 : RTC Application Software shall not use cyclic (e.g. contained in a repetitive loop) Launch Data Bus (LDB) reads, set, apply or issue commands.

All read, set, apply and issue commands are placed in an input queue of the LDB Gateway. Because of this queuing of LDB requests, an implementation using cyclic LDB command prevents the timely access of the input queues, possibly depriving other application's requests to the LDB.

9. USER DEFINED FUNCTION DESIGNATOR DEFINITION

User defined Function Designators (FD) are a general category of FDs for which the user generates the requirements. These FDs include pseudo FDs, Data Fusion FDs and Summary Data FDs. This section defines the naming convention to be used when assigning these types of FDs and the information that is required.

9.1 NAMING

User defined FDs can have a name up to ten characters in length. The naming convention that shall be followed is:

- Character 1 : N - Pseudo FD
  ? - for Fusion FD
  ? - for Summary Data
- Characters 2 - 4 : The three character CSCI name associated with the FD (e.g. HMF, OMS, MEQ, APU)
- Characters 5 -10 : A unique identifier (within the CSCI). This can be a sequential string of numbers or an alphanumeric description of the FD. The characters are limited to A- Z and 0 - 9. No special characters are permitted.

9.2 REQUIRED INFORMATION

The following information must be supplied when requesting a new user defined FD:

FDName :	As named above, coordinated with the Bank group to ensure uniqueness
Nomenclature :	limited to 34 characters
State :	for discretes (e.g. ON/OFF)
Units :	for analogs (e.g. DEGF)
Type :	e.g., Pseudo Discrete (PD), Pseudo (PA), etc.
SubType :	for discretes (e.g. BD, HEX)
Logical One :	for discretes (which state is by 1)
Responsible System (RSYS) :	the three character CSCI name (e.g. APU)
Fusion Algorithm :	Description of Algorithm (for Fusion FDs only)
Enumerated Class Name :	Unique name associated with the enumeration class
Enumerations :	Unique name/value pairs if specifying an FD with a new Enumerated Class

Appendix A VALID Computer Software Configuration Item Values

The following are the only valid Compute Software Configuration Item (CSCI) values to be used for RTC Application Software development. The official list is available from the CLCS web site (http://clcs.ksc.nasa.gov/sei/csci.html). The following list is provided for convenience and does not supersede the official list.

Acronym	***System***
apu	Orbiter Auxiliary Power Unit
arm	Swing Arm
bap	SRB Auxiliary Power Unit
bhy	SRB Hydraulics
cas	Common Application Support
com	Communication
cme	Main Engine Avionics
dps	Data Processing System
ecl	Environmental Control and Life Support
ecs	Environmental Control System
epd	Electrical Power, Distribution and Control
efc	Electronic Flight Controls
gls	Ground Launch Sequencer
hyd	Orbiter Hydraulics
hmf	Hypergolic Maintenance Facility
hws	Hazardous Warning System
int	Integrated Operations
ice	Surface Ice
ins	Instrumentation
kub	KU-Band Radar
lh2	Liquid Hydrogen
lo2	Liquid Oxygen
meq	Mechanisms
mps	Main Propulsion System
mst	Master
nav	Navigation
oms	Orbiter Maneuvering System / Reaction Control System
ple	Payload Test
pay	CITE (Payload Contractor)
fcp	Power Reactant Storage & Distribution/Fuel Cell
brs	Range Safety
sme	Space Shuttle Main Engines
slt	Super Lightweight Tank
wat	Firex Sound Suppression (SFOC)
cin	CCS Integration
cms	CCS Master
wtr	CCS Water
pwr	CCS 60Hz Power
pnu	CCS Pneumatics
hvc	CCS Heating, Venting and Air Conditioning

Appendix B CODING SOURCE FILE TEMPLATES

INTERFACE DEFINITION LANGUAGE (IDL) TEMPLATE

```
/*************************************************************************
** MODULE : <MODULE_NAME>.idl                                           **
**                                                                      **
** OVERVIEW : This example is intended to illustrate the structure and  **
**            format of an IDL file.                                    **
**                                                                      **
** NOTES: The copyright and block header comment are required for each  **
**        IDL file.                                                     **
**                                                                      **
**   REVISION HISTORY                                                   **
**  -----------------                                                   **
**        Rev          Description                                     **
**                     Author                                          **
**                     Date                                            **
**                                                                      **
**   Copyright 1998  National Aeronautics and Space Administration     **
**                   All Rights Reserved                                **
*************************************************************************/

#ifndef <MODULE_NAME>_IDL
#define <MODULE_NAME>_IDL

module <MODULE_NAME>
{
    // Exceptions:

    // Constants:

    //******************************************************
    //**  <BASE> - base interface for derived interfaces  **
    //******************************************************
    interface <BASE>
    {
        // Exceptions:

        // Constants:

        // Methods:

    };  // end interface <BASE>

    //*****************************************************************
    //**  <DERIVED> - an interface derived from <BASE>.  Interfaces can **
    //**              be inherited but there is no explicit overloading **
    //**              or overwritting allowed within the IDL.           **
    //*****************************************************************
    interface <DERIVED> : <BASE>
    {
        // Exceptions:

        // Constants:

        // Methods:

    };  // end interface <DERIVED>
```

```
};    // end module <MODULE_NAME>

#endif // <MODULE_NAME>_IDL
```

C++ Header TEMPLATE

```
/*************************************************************************
** MODULE : <MODULE_NAME>.h                                             **
**                                                                      **
** OVERVIEW : This example is intended to illustrate the structure and  **
**            format of an IDL file.                                    **
**                                                                      **
** NOTES: The copyright and block header comment are required for each  **
**        IDL file.                                                     **
**                                                                      **
**   REVISION HISTORY                                                   **
**  -----------------                                                   **
**        Rev          Description                                     **
**                     Author                                          **
**                     Date                                            **
**                                                                      **
**   Copyright 1998  National Aeronautics and Space Administration     **
**                   All Rights Reserved                                **
*************************************************************************/

#ifndef <MODULE_NAME>_H
#define <MODULE_NAME>_H

//--  system include files  (i.e. <iostream.h>)

//--  CLCS include files  (i.e. asv_fdsFunctionDesignator.h)

class <class_name>
{
    public :

    protected :

    private :
} ;

#endif  // <MODULE_NAME>_H
```

C++ Source File TEMPLATE

```
/*************************************************************************
** MODULE : <MODULE_NAME>.C                                             **
**                                                                      **
** OVERVIEW : This example is intended to illustrate the structure and  **
**            format of an IDL file.                                    **
**                                                                      **
** NOTES: The copyright and block header comment are required for each  **
**        IDL file.                                                     **
**                                                                      **
**   REVISION HISTORY                                                   **
**  -----------------                                                   **
**        Rev          Description                                     **
**                     Author                                          **
**                     Date                                            **
**                                                                      **
**   Copyright 1998  National Aeronautics and Space Administration     **
**                   All Rights Reserved                                **
*************************************************************************/

//--  system include files (e.g. <iostream.h>

//--  CLCS include files  (e.g. "<MODULE_NAME>.h)

//--  Global Constants

//--  External Declarations

//--  Class method definitions  --


Class_Name::Class_Name (...)
{
}


Class_Name::~Class_Name (void)
{
}

...
```